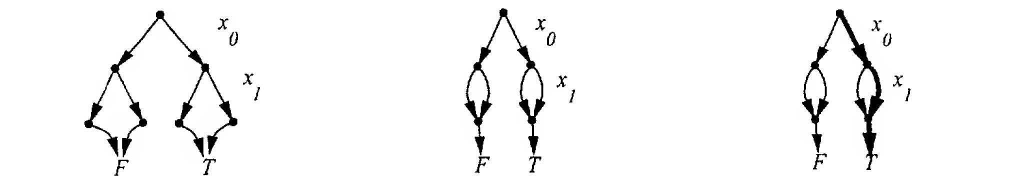
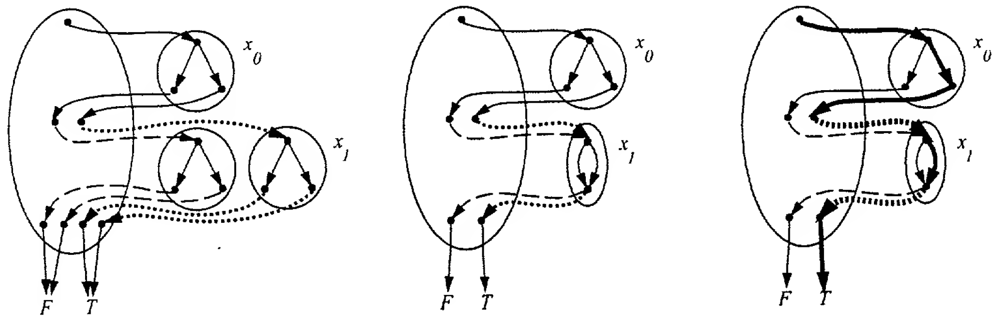
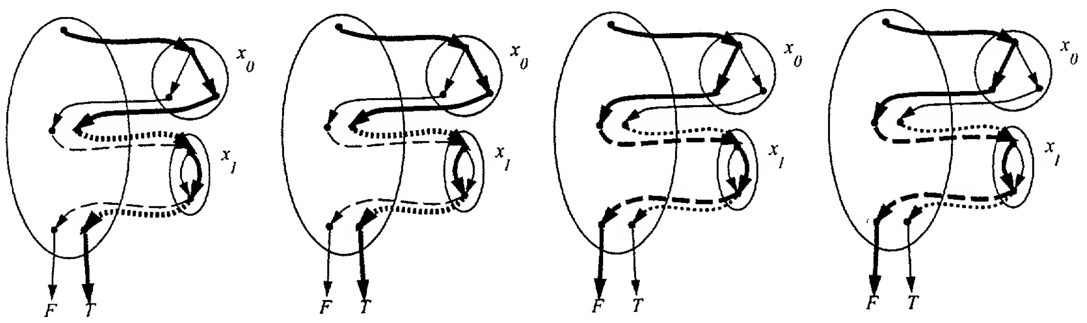


(a) Decision tree for $\lambda x_0 x_1 . x_0$ (b) Folded form (OBDD) (c) Path corresponding to the assignment $[x_0 \mapsto T, x_1 \mapsto T]$

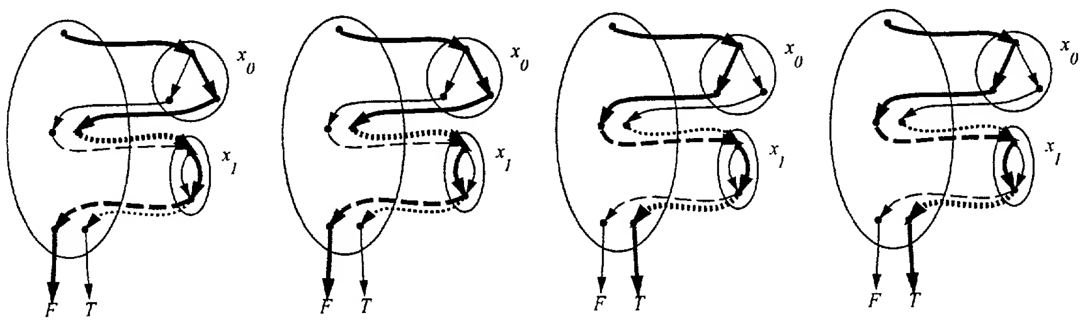


(d) Fully expanded form (e) Folded form (CFLOBDD) (f) Path corresponding to the assignment $[x_0 \mapsto T, x_1 \mapsto T]$

Figure 1:

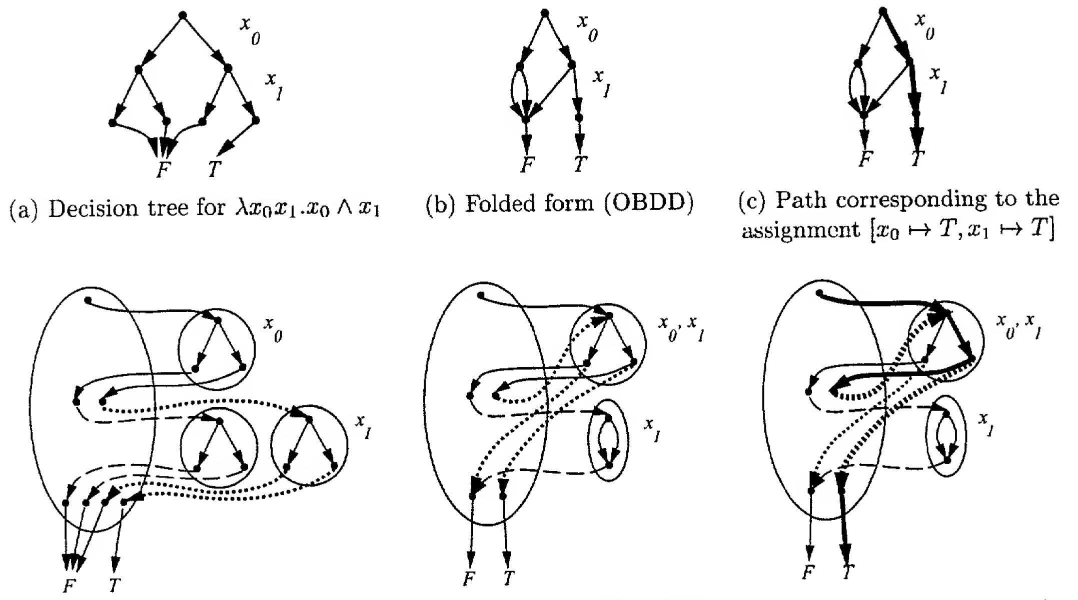


(a) $[x_0 \mapsto T, x_1 \mapsto T]$ (b) $[x_0 \mapsto T, x_1 \mapsto F]$ (c) $[x_0 \mapsto F, x_1 \mapsto T]$ (d) $[x_0 \mapsto F, x_1 \mapsto F]$



(e) (f) (g) (h)

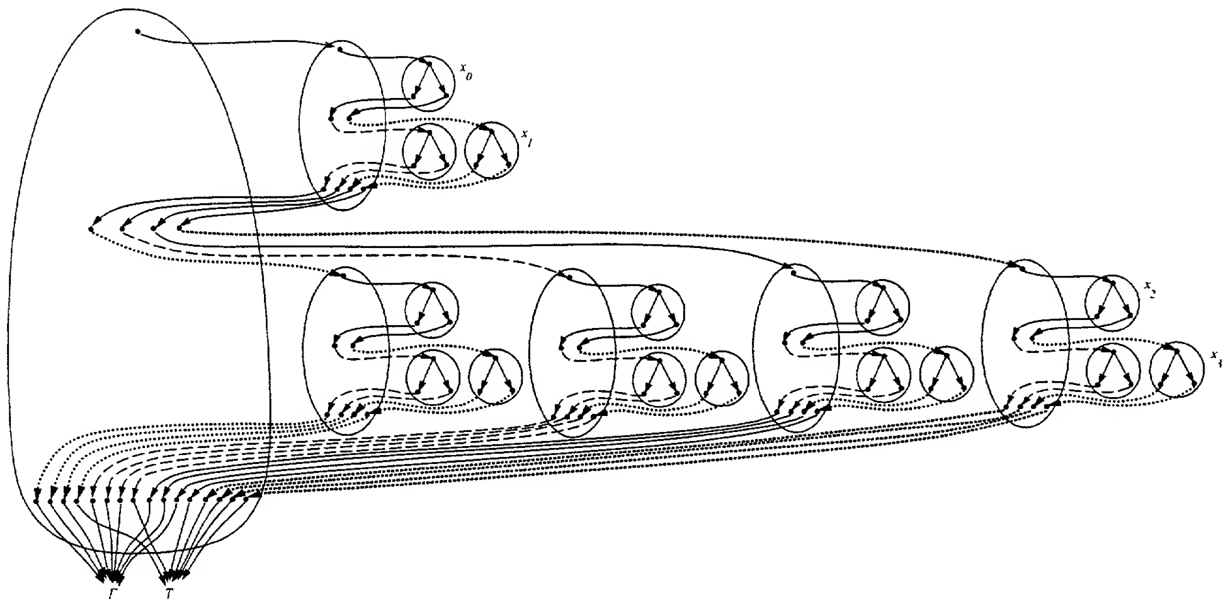
Figure 2:



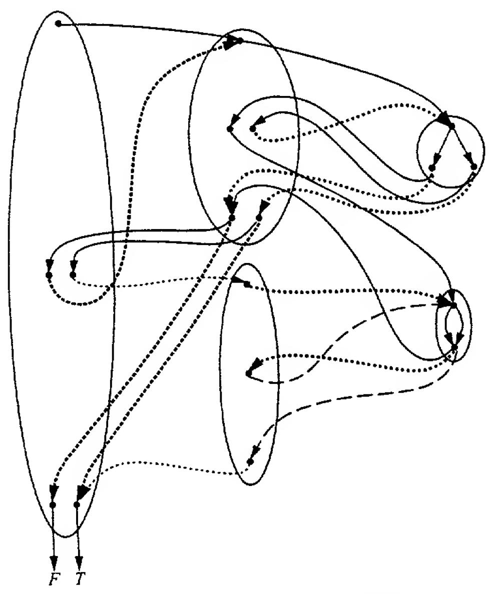
(a) Decision tree for $\lambda x_0 x_1 . x_0 \wedge x_1$ (b) Folded form (OBDD) (c) Path corresponding to the assignment $[x_0 \mapsto T, x_1 \mapsto T]$

(d) Fully expanded form (e) Folded form (CFLOBDD) (f) Path corresponding to the assignment $[x_0 \mapsto T, x_1 \mapsto T]$

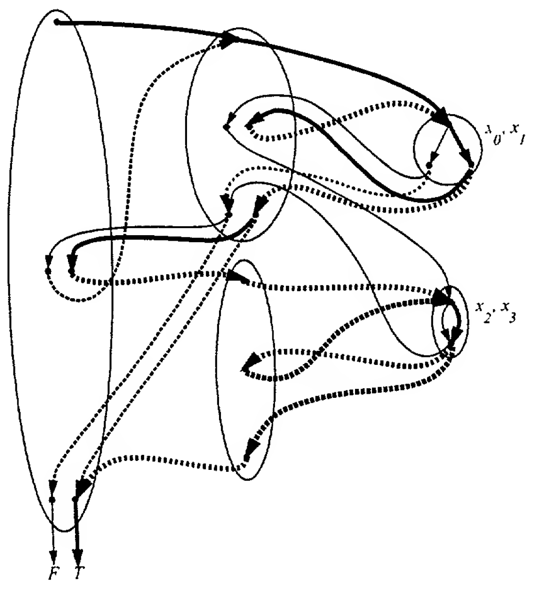
Figure 3:



(a) Fully expanded form for $\lambda x_0 x_1 x_2 x_3 . (x_0 \wedge x_1) \vee (x_2 \wedge x_3)$

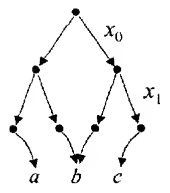


(b) Folded form (CFLOBDD)

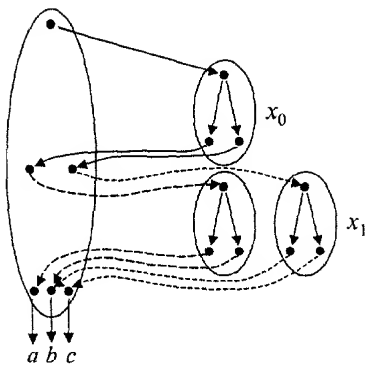


(c) Path corresponding to the assignment $[x_0 \mapsto T, x_1 \mapsto T, x_2 \mapsto T, x_3 \mapsto T]$

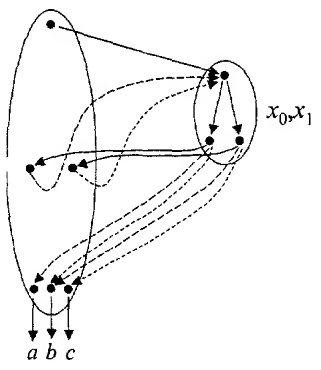
Figure 4:



(a) Decision tree for a function of type Bool × Bool → $\{a, b, c\}$

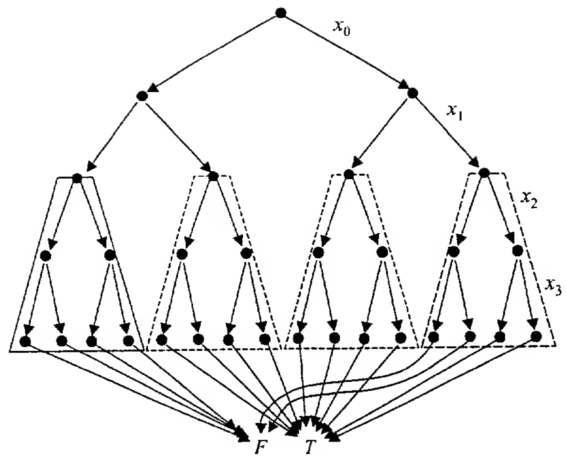


(b) Fully expanded form

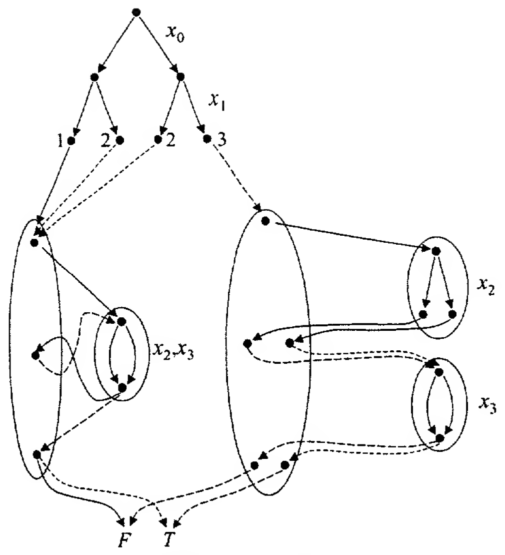


(c) Multi-terminal CFLOBDD for decision-tree (a)

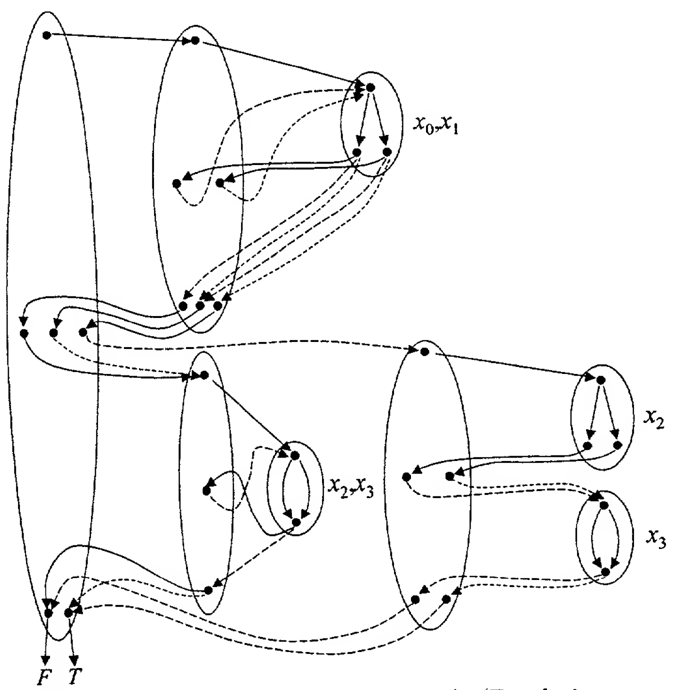
Figure 5:



(a) Decision tree for the function $\lambda x_0 x_1 x_2 x_3.(x_0 \oplus x_1) \vee (x_0 \wedge x_1 \wedge x_2)$.

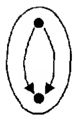


(b) Hybrid of decision tree for x_0 and x_1, and CFLOBDDs for x_2 and x_3, for the function $\lambda x_0 x_1 x_2 x_3.(x_0 \oplus x_1) \vee (x_0 \wedge x_1 \wedge x_2)$. The solid, dotted, and dashed edges from the four vertices labeled 1, 2, 2, and 3, respectively, correspond to the solid, dotted, and dashed trapezoids in (a).

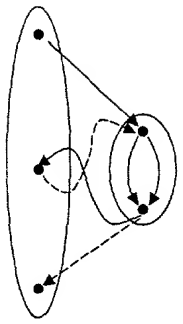


(c) CFLOBDD for the function $\lambda x_0 x_1 x_2 x_3.(x_0 \oplus x_1) \vee (x_0 \wedge x_1 \wedge x_2)$. (For clarity, some of the level-0 groupings have been duplicated.) The dotted and dashed edges in the lower half of the diagram correspond to the analogous edges in (b).

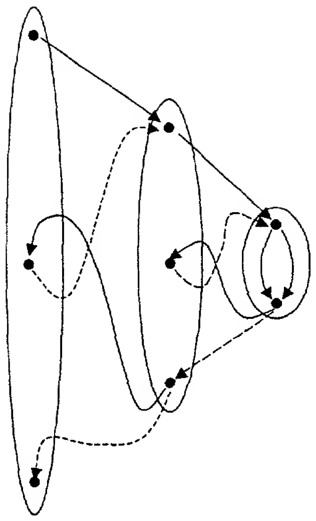
Figure 6:



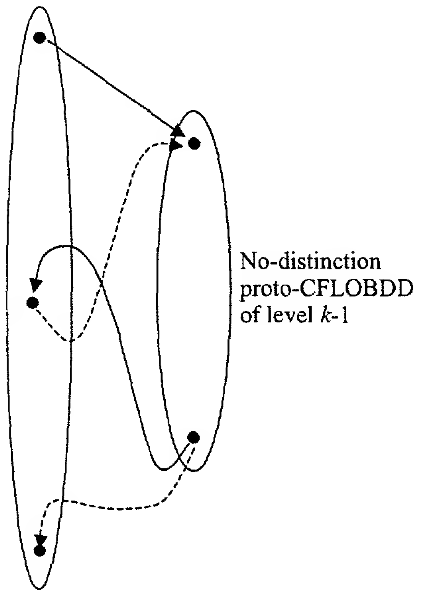
(a) The no-distinction proto-CFLOBDD of level 0.



(b) The no-distinction proto-CFLOBDD of level 1.

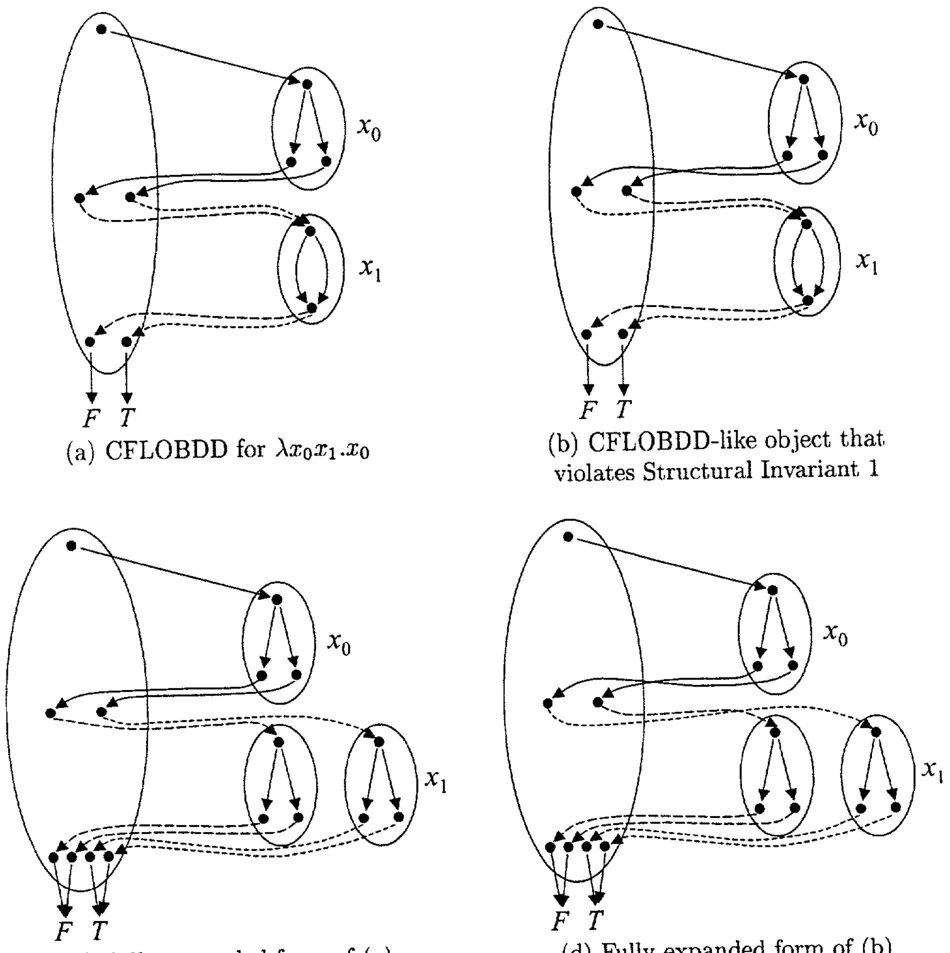


(c) The no-distinction proto-CFLOBDD of level 2.



(d) Schematic drawing showing how the no-distinction proto-CFLOBDD of level k is constructed from the no-distinction proto-CFLOBDD of level $k-1$

Figure 7:



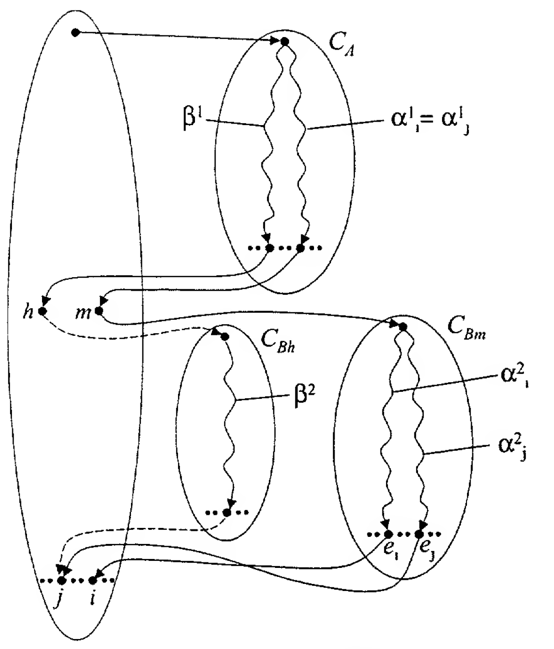
(a) CFLOBDD for $\lambda x_0 x_1 . x_0$

(b) CFLOBDD-like object that violates Structural Invariant 1

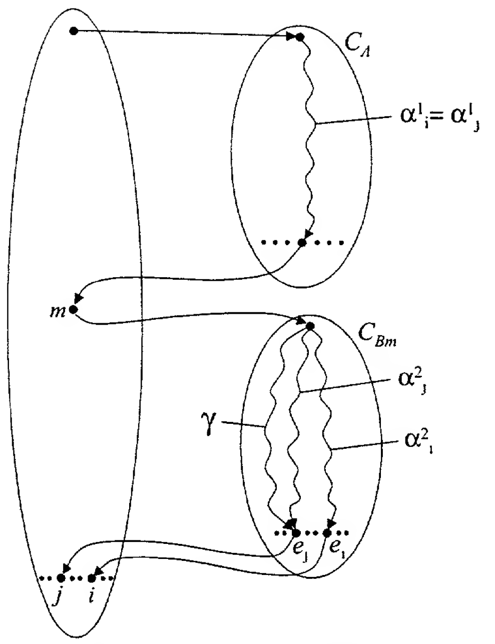
(c) Fully expanded form of (a)

(d) Fully expanded form of (b)

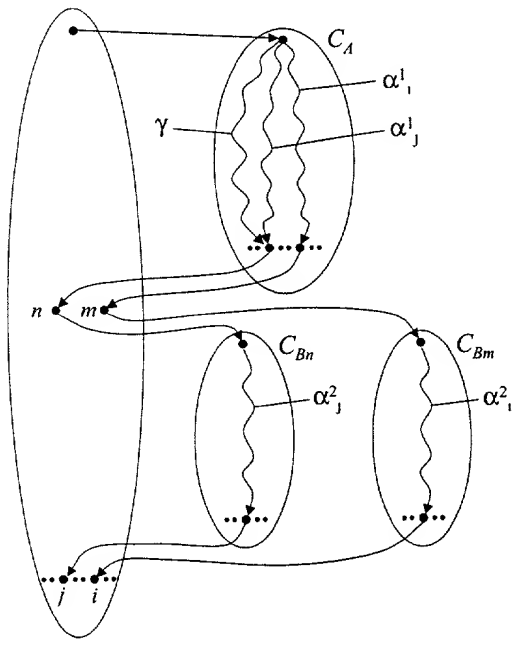
Figure 8:



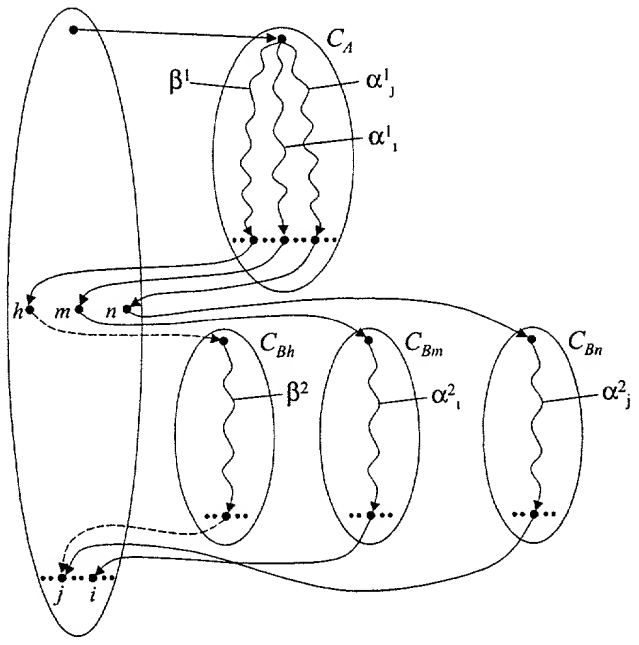
(a) Case 1.A of Proposition 1.



(b) Case 1.B of Proposition 1.



(c) Case 2.A of Proposition 1.



(d) Case 2.B of Proposition 1.

Figure 9:

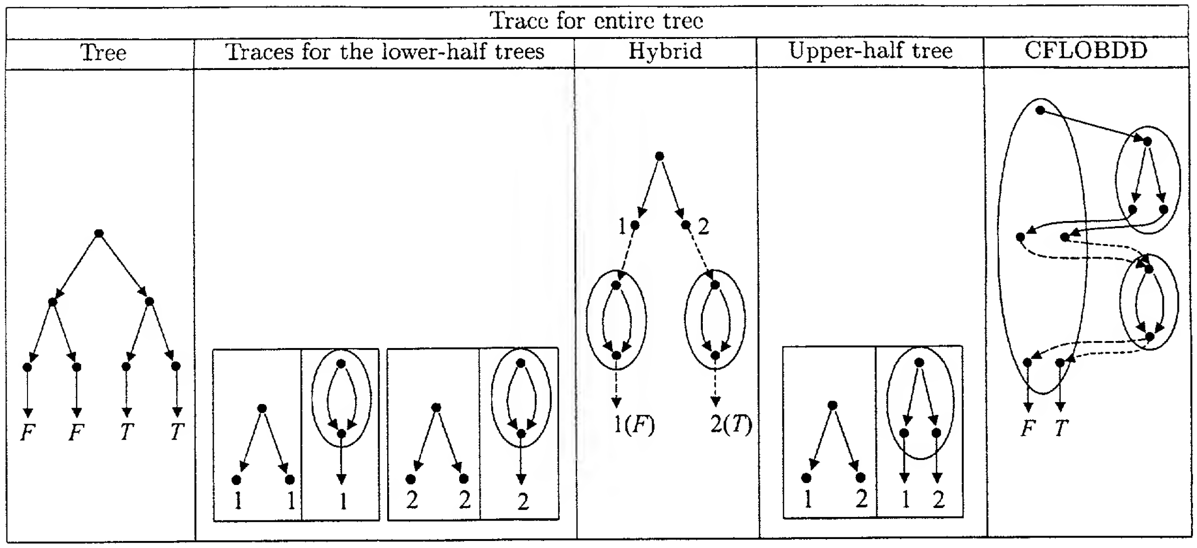


Figure 10: The *Fold* trace generated by the application of Algorithm 1 to the decision tree shown in Figure 1(a) to create the CFLOBDD shown in Figure 1(e).

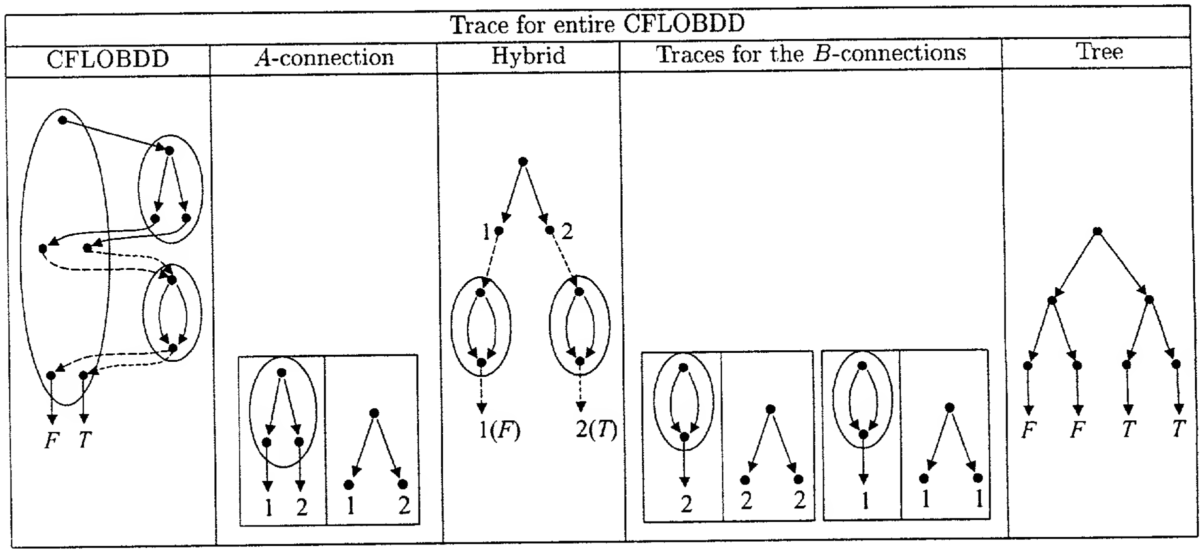


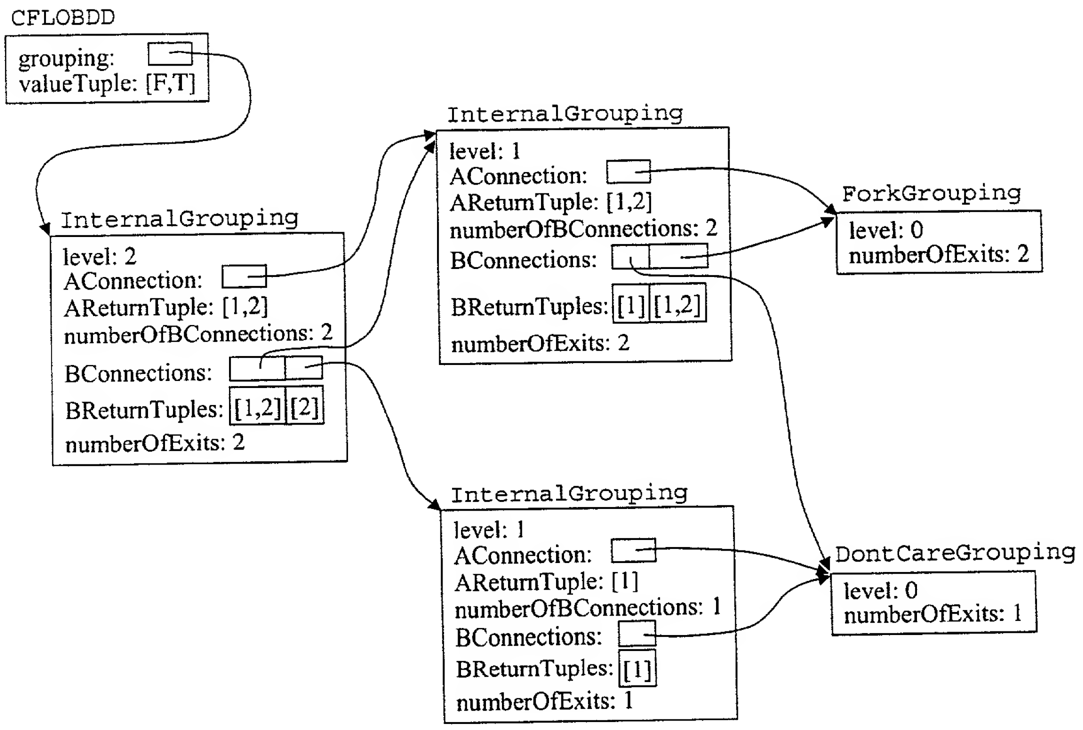
Figure 11: The *Unfold* trace generated by the application of *Unfold* to the CFLOBDD shown in Figure 1(e) to create the decision tree shown in Figure 1(a).

```
abstract class Grouping {
  level: int
  numberOfExits: int
}

class InternalGrouping extends Grouping {
  AConnection: Grouping
  AReturnTuple: ReturnTuple
  numberOfBConnections: int
  BConnections: array[1..numberOfBConnections] of Grouping
  BReturnTuples: array[1..numberOfBConnections] of ReturnTuple
}

class DontCareGrouping extends Grouping {
  level = 0
  numberOfExits = 1
}

class ForkGrouping extends Grouping {
  level = 0
  numberOfExits = 2
}

class CFLOBDD { // Multi-terminal CFLOBDD
  grouping: Grouping
  valueTuple: ValueTuple
}
```

Figure 12:

$\{x_1, \ldots, x_n\}$	a set consisting of the elements $x_1, \ldots, x_n$
$[x_1, \ldots, x_n]$	a tuple consisting of the elements $x_1, \ldots, x_n$, in order
$\{[x_1, y_1], \ldots, [x_n, y_n]\}$	a map in which x_1 maps to y_1, ..., x_n maps to y_n
$\{\}$	the empty set
$[]$	the empty tuple
$T\|\|t$	the tuple formed by appending t to the right end of tuple T
$SplitOnLast(T)$	the pair $[U, v]$, where $U\|\|v = T$
$\|S\|, \|T\|$	the number of elements in set S and tuple T, respectively
$T(i)$	the i^{th} element of tuple T
$[i..j]$, where i, j are integers	a tuple consisting of the elements $i, i+1, i+2, \ldots, j$, in order
$\{exp : iterator\}$	set former
$[exp : iterator]$	tuple former

Figure 13:

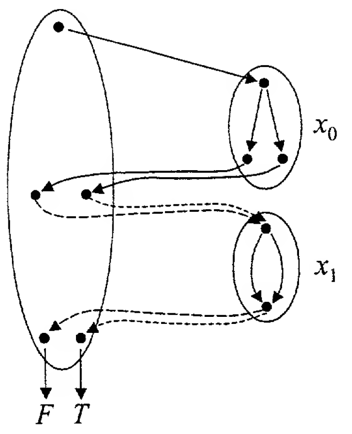


The CFLOBDD from Figure 4(b) depicted as an instance of the class `CFLOBDD` defined in Figure 12.

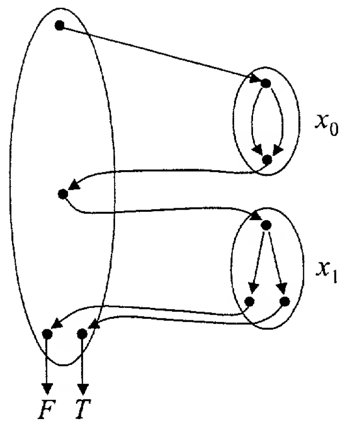
Figure 14:

```
CFLOBDD ConstantCFLOBDD(int k, value v) {
  return RepresentativeCFLOBDD(NoDistinctionProtoCFLOBDD(k),[v])
}

CFLOBDD FalseCFLOBDD(int k) {
  return ConstantCFLOBDD(k,F)
}

CFLOBDD TrueCFLOBDD(int k) {
  return ConstantCFLOBDD(k,T)
}

InternalGrouping NoDistinctionProtoCFLOBDD(int k) {
  if (k == 0) return RepresentativeDontCareGrouping
  InternalGrouping g = new InternalGrouping(k)
  g.AConnection = NoDistinctionProtoCFLOBDD(k-1)
  g.AReturnTuple = [1]
  g.numberOfBConnections = 1
  g.BConnections[1] = g.AConnection
  g.BReturnTuples[1] = [1]
  g.numberOfExits = 1
  return RepresentativeGrouping(g)
}
```

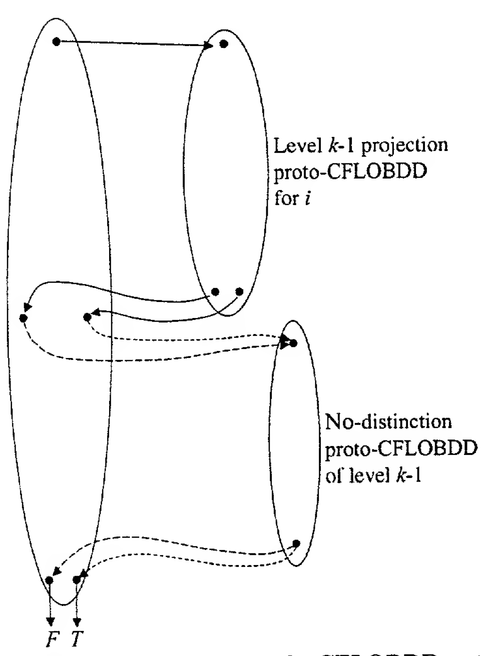
Figure 15:



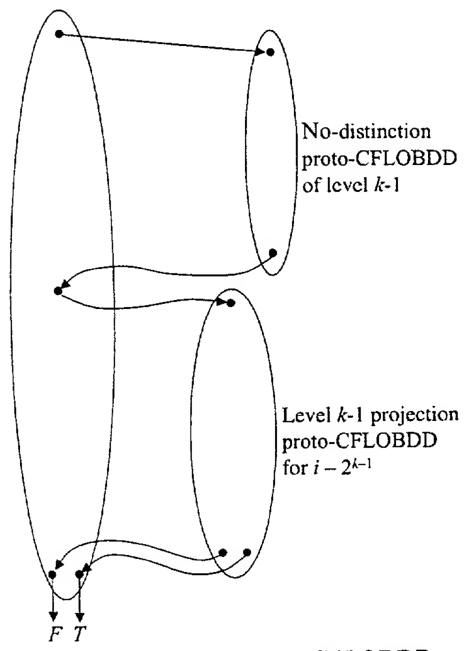
(a) CFLOBDD for $\lambda x_0 x_1 . x_0$



(b) CFLOBDD for $\lambda x_0 x_1 . x_1$



(c) Schematic drawing of CFLOBDDs that represent projection functions of the form $\lambda x_0, x_1, \ldots, x_{2^k-1}.x_i$, when $0 \leq i < 2^{k-1}$.

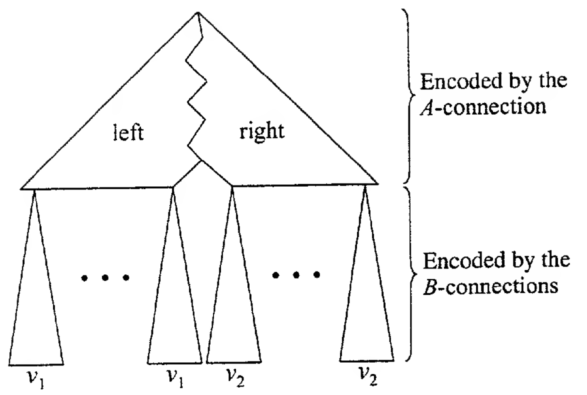


(d) Schematic drawing of CFLOBDDs that represent projection functions of the form $\lambda x_0, x_1, \ldots, x_{2^k-1}.x_i$, when $2^{k-1} \leq i < 2^k$.

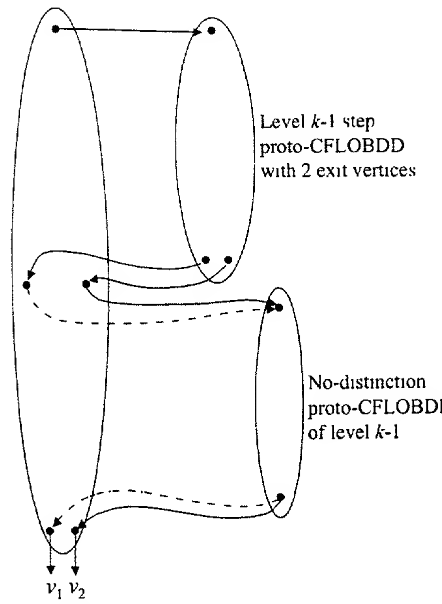
Figure 16:

```
CFLOBDD ProjectionCFLOBDD(int k, int i) {
  assert(0 <= i < 2**k)
  return RepresentativeCFLOBDD(ProjectionProtoCFLOBDD(k,i),[F,T])
}

Grouping ProjectionProtoCFLOBDD(int k, int i) {
  if (k == 0) { // i must also be 0
    return RepresentativeForkGrouping
  }
  else { // Create an appropriate InternalGrouping
    InternalGrouping g = new InternalGrouping(k)
    if (i < 2**(k-1)) { // i falls in AConnection range
      g.AConnection = ProjectionProtoCFLOBDD(k-1,i)
      g.AReturnTuple = [1,2]
      g.numBConnections = 2
      g.BConnection[1] = NoDistinctionProtoCFLOBDD(k-1)
      g.BReturnTuples[1] = [1]
      g.BConnection[2] = g.BConnection[1]
      g.BReturnTuples[2] = [2]
      g.numExits = 2
    }
    else { // i falls in BConnection range
      g.AConnection = NoDistinctionProtoCFLOBDD(k-1)
      g.AReturnTuple = [1]
      g.numBConnections = 1
      i = i - 2**(k-1)) // Remove high-order bit for recursive call
      g.BConnection[1] = ProjectionProtoCFLOBDD(k-1,i)
      g.BReturnTuples[1] = [1,2]
      g.numExits = 2
    }
    return RepresentativeGrouping(g)
  }
}
```

Figure 17:

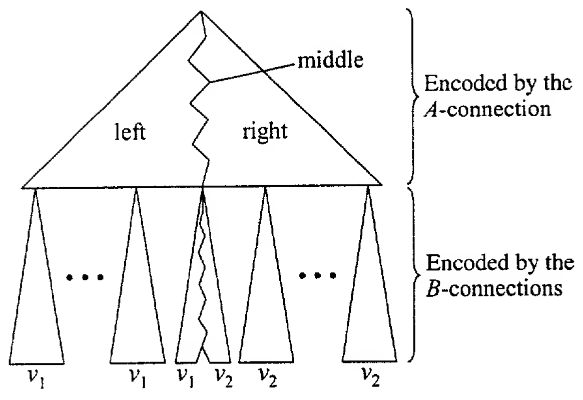


(a) Decision tree for a step function in which the step is made at a value $i = [x_0x_1 \ldots x_{2^k-1}]$ such that $i \bmod 2^{2^{k-1}} = 0$.

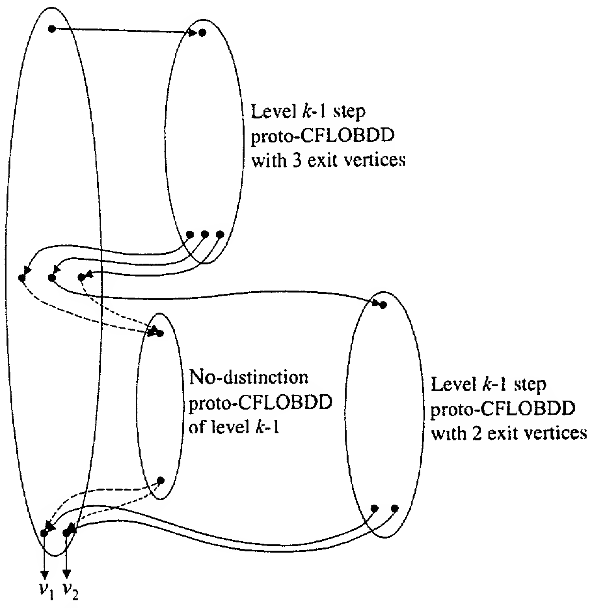


Level k-1 step proto-CFLOBDD with 2 exit vertices
No-distinction proto-CFLOBDD of level k-1
v_1 v_2

(b) Schematic drawing showing the structure of a level-k CFLOBDD for a step function in which the step is made at a value $i = [x_0x_1 \ldots x_{2^k-1}]$ such that $i \bmod 2^{2^{k-1}} = 0$.



(c) Decision tree for a step function in which the step is made at a value $i = [x_0x_1 \ldots x_{2^k-1}]$ such that $i \bmod 2^{2^{k-1}} \neq 0$.



Level k-1 step proto-CFLOBDD with 3 exit vertices
No-distinction proto-CFLOBDD of level k-1
Level k-1 step proto-CFLOBDD with 2 exit vertices
v_1 v_2

(d) Schematic drawing showing the structure of a level-k CFLOBDD for a step function in which the step is made at a value $i = [x_0x_1 \ldots x_{2^k-1}]$ such that $i \bmod 2^{2^{k-1}} \neq 0$.

Figure 18:

```
CFLOBDD StepCFLOBDD(int k, int i, value v1, value v2) {
  assert(0 <= i <= 2**(2**k))
  if (v1 == v2) return RepresentativeCFLOBDD(NoDistinctionProtoCFLOBDD(k),[v1])
  if (i == 0) return RepresentativeCFLOBDD(NoDistinctionProtoCFLOBDD(k),[v2])
  if (i == 2**(2**k)) return RepresentativeCFLOBDD(NoDistinctionProtoCFLOBDD(k),[v1])
  Grouping g = StepProtoCFLOBDD(k, i, 0, 2**(2**k) - i)
  return RepresentativeCFLOBDD(g,[v1,v2])
}

Grouping StepProtoCFLOBDD(int k, int left, int middle, int right) {
  assert(middle == 0 || middle == 1)
  assert(left + middle + right == 2**(2**k))
  if (k == 0) { // Base case
    if (left == 2 || right == 2) return RepresentativeDontCareGrouping
    else return RepresentativeForkGrouping
  }
  InternalGrouping g = new InternalGrouping(k)
  g.numberOfExits = (left != 0) + (middle != 0) + (right != 0)
  int P = 2**(2**(k-1))
  int a = left/P // a holds the 2**(k-1) most-significant bits of left
  int c = right/P // c holds the 2**(k-1) most-significant bits of right
  int b = (left mod P + right mod P)/P + middle // 0 for Fig.18(a); 1 for Fig.18(c)

  // Create the AConnection -----------------------------------------------
  int numberOfAExits = (a != 0) + (b != 0) + (c != 0) // Upto 3 exits
  g.AConnection = StepProtoCFLOBDD(k-1, a, b, c)
  g.AReturnTuple = [1..numberOfAExits]

  // Create the BConnections ----------------------------------------------
  g.numBConnections = numberOfAExits
  int curConnection = 1
  if (a != 0) {
    g.BConnection[curConnection] = NoDistinctionProtoCFLOBDD(k-1)
    g.BReturnTuples[curConnection] = [1]
    curConnection = curConnection + 1
  }
  if (b != 0) {
    int aa = left mod P
    int cc = right mod P
    int bb = middle
    g.BConnection[curConnection] = StepProtoCFLOBDD(k-1, aa, bb, cc)
    ReturnTuple rt
    if (aa != 0) rt = rt || 1 // Connect to first exit
    if (bb != 0)
      if (left != 0) rt = rt || 2
      else rt = rt || 1
    if (cc != 0) rt = rt || g.numberOfExits // Connect to last exit
    g.BReturnTuples[curConnection] = rt
    curConnection = curConnection + 1
  }
  if (c != 0) {
    g.BConnection[curConnection] = NoDistinctionProtoCFLOBDD(k-1)
    g.BReturnTuples[curConnection] = [g.numberOfExits] // Connect to last exit
  }

  return RepresentativeGrouping(g)
}
```

Figure 19:

```
CFLOBDD FlipValueTupleCFLOBDD(CFLOBDD c) {
  assert(|c.valueTuple| == 2)
  return RepresentativeCFLOBDD(c.grouping,[c.valueTuple[2],c.valueTuple[1]])
}

CFLOBDD ComplementCFLOBDD(CFLOBDD c) {
  if (c == FalseCFLOBDD(c.grouping.level)) return TrueCFLOBDD(c.grouping.level)
  if (c == TrueCFLOBDD(c.grouping.level)) return FalseCFLOBDD(c.grouping.level)
  return FlipValueTupleCFLOBDD(c)
}
```

Figure 20:

```
CFLOBDD ScalarMultiplyCFLOBDD(CFLOBDD c, Value v) {
  if (v == zero) return ConstantCFLOBDD(c.level,zero)
  return RepresentativeCFLOBDD(c.grouping,[v*c.valueTuple(i) | i ∈ [1..|c.valueTuple|])
}
```

Figure 21:

```
Tuple×Tuple CollapseClassesLeftmost(Tuple equivClasses) {
  // Project the tuple equivClasses, preserving left-to-right order,
  // retaining the leftmost instance of each class
  Tuple projectedClasses =
        [ equivClasses(i)
        : i ∈ [1..|equivClasses|]
        | i = min{j ∈ [1..|equivClasses|] | equivClasses(j) = equivClasses(i)}
        ]

  // Create tuple in which classes in equivClasses are renumbered
  // according to their ordinal position in projectedClasses
  Map orderOfProjectedClasses =
          {[x,i] : i ∈ [1..|projectedClasses|] | x = projectedClasses(i)}
  Tuple renumberedClasses = [orderOfProjectedClasses(v) : v ∈ equivClasses ]

  return [projectedClasses, renumberedClasses]
}
```

Figure 22:

```
CFLOBDD BinaryApplyAndReduce(CFLOBDD n1, CFLOBDD n2, Op op) {
  assert(n1.grouping.level == n2.grouping.level)

  // Perform cross product
  Grouping×PairTuple [g,pt] = PairProduct(n1.grouping,n2.grouping)

  // Create tuple of ''leaf'' values
  ValueTuple deducedValueTuple =
            [ op(n1.valueTuple[i1],n2.valueTuple[i2]) : [i1,i2] ∈ pt ]

  // Collapse duplicate leaf values, folding to the left
  Tuple×Tuple [inducedValueTuple,inducedReductionTuple] =
                CollapseClassesLeftmost(deducedValueTuple)

  // Perform corresponding reduction on g,
  // folding g's exit vertices w.r.t. inducedReductionTuple
  Grouping g' = Reduce(g, inducedReductionTuple)

  return RepresentativeCFLOBDD(g', inducedValueTuple)
}
```

Figure 23:

```
Grouping×PairTuple PairProduct(Grouping g1, Grouping g2) {
  if (g1 and g2 are both no-distinction proto-CFLOBDDs) return [ g1, [[1,1]] ]
  if (g1 is a no-distinction proto-CFLOBDD)
    return [ g2, [[1,k] : k ∈ [1..g2.numberOfExits]] ]
  if (g2 is a no-distinction proto-CFLOBDD)
    return [ g1, [[k,1] : k ∈ [1..g1.numberOfExits]] ]
  if (g1 and g2 are both fork groupings) return [ g1, [[1,1],[2,2]] ]

  // Pair the A-connections ------------------------------------------------------
  Grouping×PairTuple [gA,ptA] = PairProduct(g1.AConnection,
                                            g2.AConnection)

  InternalGrouping g = new InternalGrouping(g1.level)
  g.AConnection = gA
  g.AReturnTuple = [1..|ptA|] // Represents the middle vertices
  g.numberOfBConnections = |ptA|

  // Pair the B-connections, but only for pairs in ptA ---------------------------
  Tuple ptAns = [] // Descriptor of pairings of exit vertices
  for (j = 1; j <= |ptA|; j++) { // Create a B-connection for each middle vertex
    Grouping×PairTuple [gB,ptB] = PairProduct(g1.BConnections[ptA(j)(1)],
                                              g2.BConnections[ptA(j)(2)])
    g.BConnections[j] = gB
    // Now create g.BReturnTuples[j], and augment ptAns as necessary
    g.BReturnTuples[j] = []
    for (i = 1; i <= |ptB|; i++) {
      c1 = g1.BReturnTuples[ptA(j)(1)](ptB(i)(1)) // an exit vertex of g1
      c2 = g2.BReturnTuples[ptA(j)(2)](ptB(i)(2)) // an exit vertex of g2
      if([c1,c2] ∈ ptAns) { // Not a new exit vertex of g
        index = the k such that ptAns(k) == [c1,c2]
        g.BReturnTuples[j] = g.BReturnTuples[j] || index
      }
      else { // Identified a new exit vertex of g
        g.numberOfExits = g.numberOfExits + 1
        g.BReturnTuples[j] = g.BReturnTuples[j] || g.numberOfExits
        ptAns = ptAns || [c1,c2]
      }
    }
  }

  return [RepresentativeGrouping(g), ptAns]
}
```

Figure 24:

```
// Insert h,ReturnTuple as next B-connection of g, if they are a new combination;
// otherwise, return the index of the existing occurrence of h,ReturnTuple
int InsertBConnection(InternalGrouping g, Grouping h, ReturnTuple returnTuple) {
  for (i = 1; i <= g.numberOfBConnections; i++) {
    if (g.BConnections[i] == h && g.BReturnTuples[i] == returnTuple) return i
  }
  // Not found -- insert h,ReturnTuple as next B-connection of g
  g.numberOfBConnections = g.numberOfBConnections + 1
  g.BConnections[g.numberOfBConnections] = h
  g.BReturnTuples[g.numberOfBConnections] = returnTuple
  return g.numberOfBConnections
}

Grouping Reduce(Grouping g, ReductionTuple reductionTuple) {
  // Test whether any reduction actually needs to be carried out
  if (reductionTuple == [1..|reductionTuple|]) return g

  // If only one exit vertex, then collapse to no-distinction proto-CFLOBDD
  if (|{x : x ∈ reductionTuple }| == 1) return NoDistinctionProtoCFLOBDD(g.level)

  InternalGrouping g' = new InternalGrouping(g.level)
  g'.numberOfExits = |{x : x ∈ reductionTuple }|

  // Reduce each B-connection of g w.r.t. reductionTuple, while gathering
  // information in reductionTupleA about how to reduce the A-connection of g
  Tuple reductionTupleA = []
  for (i = 1; i <= g.numberOfBConnections; i++) {
    Tuple deducedReturnClasses = [ reductionTuple(v) : v ∈ g.BReturnTuples[i] ]
    Tuple×Tuple [inducedReturnTuple,inducedReductionTuple] =
                CollapseClassesLeftmost(deducedReturnClasses)

    // Perform reduction on g.BConnections[i], w.r.t. inducedReductionTuple
    Grouping h = Reduce(g.BConnection[i], inducedReductionTuple)

    // Insert h,inducedReturnTuple as next B-connection, but only if new
    int position = InsertBConnection(g', h, inducedReturnTuple)

    // Build up the tuple that indicates how to reduce the A-connection
    reductionTupleA = reductionTupleA || position
  }

  // Reduce the A-connection w.r.t. reductionTupleA
  Tuple×Tuple [inducedReturnTuple,inducedReductionTuple] =
              CollapseClassesLeftmost(reductionTupleA)

  // Perform reduction on g.AConnection, w.r.t. inducedReductionTuple
  Grouping h' = Reduce(g.AConnection,inducedReductionTuple)

  g'.AConnection = h'
  g'.AReturnTuple = inducedReturnTuple

  return RepresentativeGrouping(g')
}
```

Figure 25:

```
CFLOBDD TernaryApplyAndReduce(CFLOBDD n1, CFLOBDD n2, CFLOBDD n3, Op op) {
  assert(n1.grouping.level == n2.grouping.level && n2.grouping.level == n3.grouping.level)

  // Perform triple cross product
  Grouping×TripleTuple [g,tt] = TripleProduct(n1.grouping,n2.grouping,n3.grouping)

  // Create tuple of ``leaf'' values
  ValueTuple deducedValueTuple =
             [ op(n1.valueTuple[i1],n2.valueTuple[i2],n3.valueTuple[i3]) : [i1,i2,i3] ∈ tt ]

  // Collapse duplicate leaf values, folding to the left
  Tuple×Tuple [inducedValueTuple,inducedReductionTuple] =
               CollapseClassesLeftmost(deducedValueTuple)

  // Perform corresponding reduction on g,
  // folding g's exit vertices w.r.t. inducedReductionTuple
  Grouping g' = Reduce(g, inducedReductionTuple)

  return RepresentativeCFLOBDD(g', inducedValueTuple)
}
```

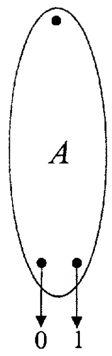
Figure 26:

```
GroupingxTripleTuple TripleProduct(Grouping g1, Grouping g2, Grouping g3) {
  if (g1, g2, and g3 are all no-distinction proto-CFLOBDDs) return [ g1, [[1,1,1]] ]
  if (g1 and g2 are no-distinction proto-CFLOBDDs)
    return [ g3, [[1,1,k] : k ∈ [1..g3.numberOfExits]] ]
  if (g1 and g3 are no-distinction proto-CFLOBDDs)
    return [ g2, [[1,k,1] : k ∈ [1..g2.numberOfExits]] ]
  if (g2 and g3 are no-distinction proto-CFLOBDDs)
    return [ g1, [[k,1,1] : k ∈ [1..g1.numberOfExits]] ]
  if (g1 is a no-distinction proto-CFLOBDD) {
    GroupingxPairTuple [g,pt] = PairProduct(g2,g3)
    return [ g, [[1,j,k] : [j,k] ∈ pt] ]
  }
  if (g2 is a no-distinction proto-CFLOBDD) {
    GroupingxPairTuple [g,pt] = PairProduct(g1,g3)
    return [ g, [[j,1,k] : [j,k] ∈ pt] ]
  }
  if (g3 is a no-distinction proto-CFLOBDD) {
    GroupingxPairTuple [g,pt] = PairProduct(g1,g2)
    return [ g, [[j,k,1] : [j,k] ∈ pt] ]
  }
  if (g1, g2, and g3 are all fork groupings) return [ g1, [[1,1,1],[2,2,2]] ]

  // Combine the A-connections -----------------------------------------------
  GroupingxTripleTuple [gA,ttA] = TripleProduct(g1.AConnection,
                                                 g2.AConnection,
                                                 g3.AConnection)

  InternalGrouping g = new InternalGrouping(g1.level)
  g.AConnection = gA
  g.AReturnTuple = [1..|ttA|] // Represents the middle vertices
  g.numberOfBConnections = |ttA|

  // Combine the B-connections, but only for triples in ttA -----------------
  Tuple ttAns = [] // Descriptor of triples of exit vertices
  for (j = 1; j <= |ttA|; j++) { // Create a B-connection for each middle vertex
    GroupingxTripleTuple [gB,ttB] = TripleProduct(g1.BConnections[ttA(j)(1)],
                                                   g2.BConnections[ttA(j)(2)],
                                                   g3.BConnections[ttA(j)(3)])
    g.BConnections[j] = gB
    // Now create g.BReturnTuples[j], and augment ttAns as necessary
    g.BReturnTuples[j] = []
    for (i = 1; i <= |ttB|; i++) {
       c1 = g1.BReturnTuples[ttA(j)(1)](ttB(i)(1)) // an exit vertex of g1
       c2 = g2.BReturnTuples[ttA(j)(2)](ttB(i)(2)) // an exit vertex of g2
       c3 = g3.BReturnTuples[ttA(j)(3)](ttB(i)(3)) // an exit vertex of g3
       if([c1,c2,c3] ∈ ttAns) { // Not a new exit vertex of g
         index = the k such that ttAns(k) == [c1,c2,c3]
         g.BReturnTuples[j] = g.BReturnTuples[j] || index
       }
       else { // Identified a new exit vertex of g
         g.numberOfExits = g.numberOfExits + 1
         g.BReturnTuples[j] = g.BReturnTuples[j] || g.numberOfExits
         ttAns = ttAns || [c1,c2,c3]
       }
    }
  }

  return [RepresentativeGrouping(g), ttAns]
}
```

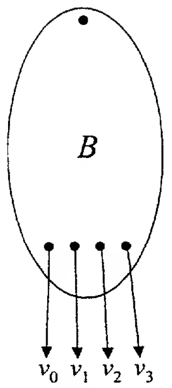
Figure 27:

Truth Table	Defining Expression	Definition Using ITE
$\begin{array}{cccc} & & \multicolumn{2}{c}{b} \\ & & F & T \\ a & F & F & F \\ & T & F & F \end{array}$	$\lambda a, b.F$	$\lambda a, b.F$
$\begin{array}{cccc} & & \multicolumn{2}{c}{b} \\ & & F & T \\ a & F & F & F \\ & T & F & T \end{array}$	$\lambda a, b.a \wedge b$	$\lambda a, b.\text{ITE}(a, b, F)$
$\begin{array}{cccc} & & \multicolumn{2}{c}{b} \\ & & F & T \\ a & F & F & F \\ & T & T & F \end{array}$	$\lambda a, b.a \wedge \neg b$	$\lambda a, b.\text{ITE}(a, \neg b, F)$
$\begin{array}{cccc} & & \multicolumn{2}{c}{b} \\ & & F & T \\ a & F & F & F \\ & T & T & T \end{array}$	$\lambda a, b.a$	$\lambda a, b.a$
$\begin{array}{cccc} & & \multicolumn{2}{c}{b} \\ & & F & T \\ a & F & F & T \\ & T & F & F \end{array}$	$\lambda a, b.\neg a \wedge b$	$\lambda a, b.\text{ITE}(a, F, b)$
$\begin{array}{cccc} & & \multicolumn{2}{c}{b} \\ & & F & T \\ a & F & F & T \\ & T & F & T \end{array}$	$\lambda a, b.b$	$\lambda a, b.b$
$\begin{array}{cccc} & & \multicolumn{2}{c}{b} \\ & & F & T \\ a & F & F & T \\ & T & T & F \end{array}$	$\lambda a, b.a \oplus b$	$\lambda a, b.\text{ITE}(a, \neg b, b)$
$\begin{array}{cccc} & & \multicolumn{2}{c}{b} \\ & & F & T \\ a & F & F & T \\ & T & T & T \end{array}$	$\lambda a, b.a \vee b$	$\lambda a, b.\text{ITE}(a, T, b)$
$\begin{array}{cccc} & & \multicolumn{2}{c}{b} \\ & & F & T \\ a & F & T & F \\ & T & F & F \end{array}$	$\lambda a, b.\neg(a \vee b)$	$\lambda a, b.\text{ITE}(a, F, \neg b)$
$\begin{array}{cccc} & & \multicolumn{2}{c}{b} \\ & & F & T \\ a & F & T & F \\ & T & F & T \end{array}$	$\lambda a, b.\neg(a \oplus b)$	$\lambda a, b.\text{ITE}(a, b, \neg b)$
$\begin{array}{cccc} & & \multicolumn{2}{c}{b} \\ & & F & T \\ a & F & T & F \\ & T & T & F \end{array}$	$\lambda a, b.\neg b$	$\lambda a, b.\text{ITE}(b, F, T)$
$\begin{array}{cccc} & & \multicolumn{2}{c}{b} \\ & & F & T \\ a & F & T & F \\ & T & T & T \end{array}$	$\lambda a, b.a \vee \neg b$	$\lambda a, b.\text{ITE}(a, T, \neg b)$
$\begin{array}{cccc} & & \multicolumn{2}{c}{b} \\ & & F & T \\ a & F & T & T \\ & T & F & F \end{array}$	$\lambda a, b.\neg a$	$\lambda a, b.\text{ITE}(a, F, T)$
$\begin{array}{cccc} & & \multicolumn{2}{c}{b} \\ & & F & T \\ a & F & T & T \\ & T & F & T \end{array}$	$\lambda a, b.\neg a \vee b$	$\lambda a, b.\text{ITE}(a, b, T)$
$\begin{array}{cccc} & & \multicolumn{2}{c}{b} \\ & & F & T \\ a & F & T & T \\ & T & T & F \end{array}$	$\lambda a, b.\neg(a \wedge b)$	$\lambda a, b.\text{ITE}(a, \neg b, T)$
$\begin{array}{cccc} & & \multicolumn{2}{c}{b} \\ & & F & T \\ a & F & T & T \\ & T & T & T \end{array}$	$\lambda a, b.T$	$\lambda a, b.T$

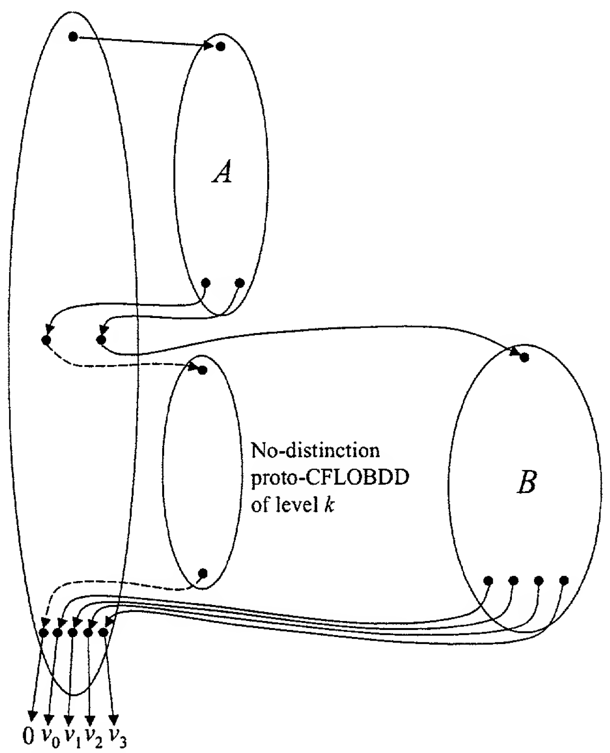
Figure 28:



(a) Level-k CFLOBDD for an array A whose elements are drawn from $\{0, 1\}$. (Schematic drawing only—details of lower-level groupings are not shown.)

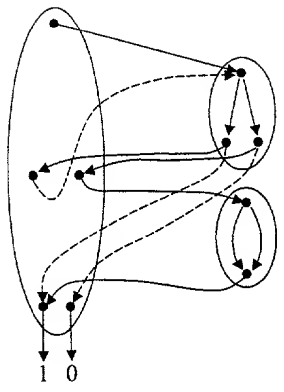


(b) Level-k CFLOBDD for an array B whose elements are drawn from $\{v_0, v_1, v_2, v_3\}$. (Schematic drawing only—details of lower-level groupings are not shown.)



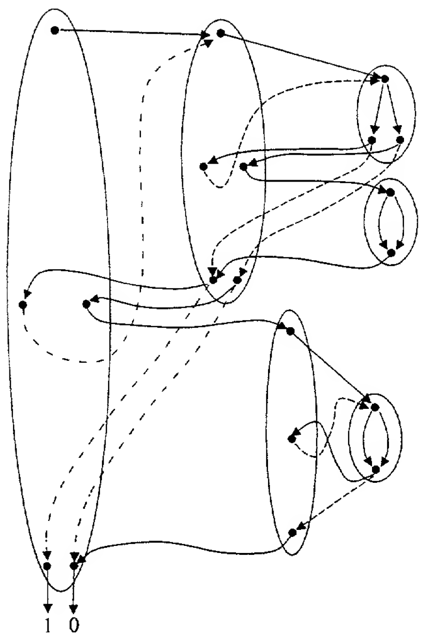
(c) Level $k+1$ CFLOBDD for array $A \otimes B$. (Schematic drawing only—details of lower-level groupings are not shown.)

Figure 29:



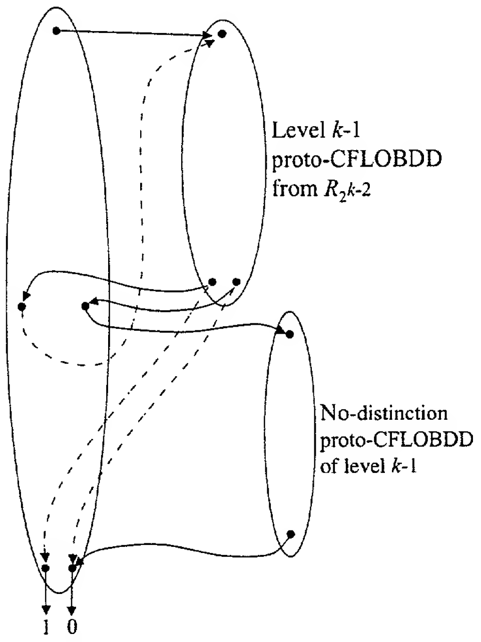
$$R_1 = \begin{array}{c|cc|c} & 0 & 1 & y_0 \\ \hline 0 & 1 & 0 & \\ 1 & 1 & 1 & \\ \hline x_0 & & & \end{array}$$

(a) Level-1 CFLOBDD for the 2 × 2 Reed-Muller transform matrix R_1, under the interleaved variable ordering.



$$R_2 = \begin{array}{cc|cc|cc|c} & & 0 & 0 & 1 & 1 & y_0 \\ & & 0 & 1 & 0 & 1 & y_1 \\ \hline 0 & 0 & 1 & 0 & 0 & 0 & \\ 0 & 1 & 1 & 1 & 0 & 0 & \\ \hline 1 & 0 & 1 & 0 & 1 & 0 & \\ 1 & 1 & 1 & 1 & 1 & 1 & \\ \hline x_0 & x_1 & & & & & \end{array}$$

(b) Level-2 CFLOBDD for the 4 × 4 Reed-Muller transform matrix R_2, under the interleaved variable ordering.



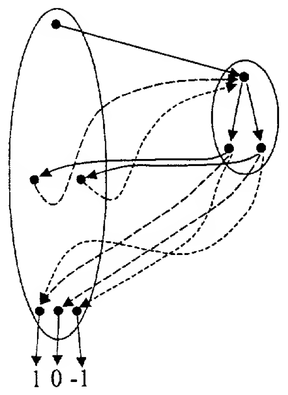
Level k-1 proto-CFLOBDD from $R_{2^{k-2}}$
No-distinction proto-CFLOBDD of level k-1
1
0

(c) Schematic drawing showing how, under the interleaved variable ordering, the level-k CFLOBDD for the $2^{2^{k-1}} \times 2^{2^{k-1}}$ Reed-Muller transform matrix $R_{2^{k-1}} = R_{2^{k-2}} \otimes R_{2^{k-2}}$ is constructed from two level $k-1$ proto-CFLOBDDs.

Figure 30:

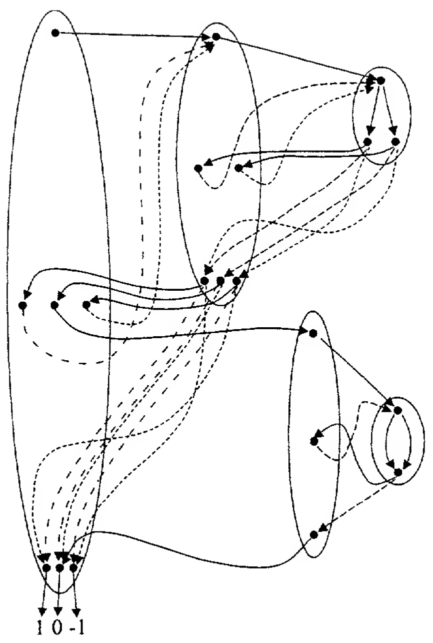
```
CFLOBDD ReedMullerCFLOBDD(int i) {
  return RepresentativeCFLOBDD(ReedMullerProtoCFLOBDD(i),[1,0])
}

InternalGrouping ReedMullerProtoCFLOBDD(int i) {
  InternalGrouping g = new InternalGrouping(i)
  if (i == 1) {
    g.AConnection = RepresentativeForkGrouping
    g.AReturnTuple = [1,2]
    g.numberOfBConnections = 2
    g.BConnections[1] = RepresentativeForkGrouping
    g.BReturnTuples[1] = [1,2]
    g.BConnections[2] = RepresentativeDontCareGrouping
    g.BReturnTuples[2] = [1]
  }
  else {
    g.AConnection = ReedMullerProtoCFLOBDD(i-1)
    g.AReturnTuple = [1,2]
    g.numberOfBConnections = 2
    g.BConnections[1] = g.AConnection
    g.BReturnTuples[1] = [1,2]
    g.BConnections[2] = NoDistinctionProtoCFLOBDD(i-1)
    g.BReturnTuples[2] = [2]
  }
  g.numberOfExits = 2
  return RepresentativeGrouping(g)
}
```

Figure 31:



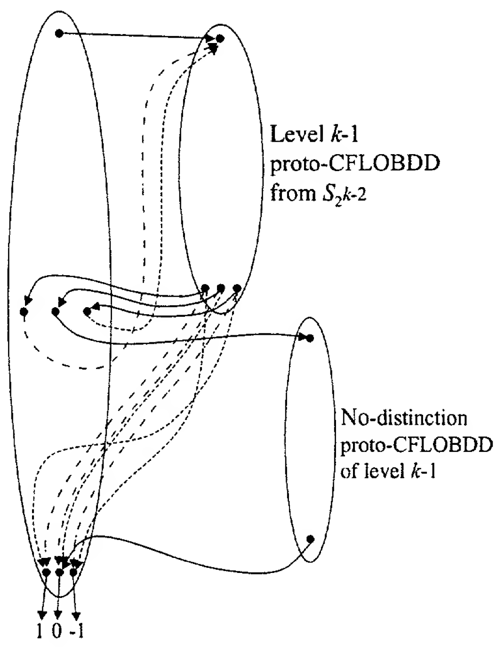
$$S_1 = \begin{array}{c|cc|l} & 0 & 1 & y_0 \\ \hline 0 & 1 & 0 & \\ 1 & -1 & 1 & \\ \hline x_0 & & & \end{array}$$

(a) Level-1 CFLOBDD for the 2 × 2 inverse Reed-Muller transform matrix S_1, under the interleaved variable ordering.



$$S_2 = \begin{array}{cc|cc|cc|l} & & 0 & 0 & 1 & 1 & y_0 \\ & & 0 & 1 & 0 & 1 & y_1 \\ \hline 0 & 0 & 1 & 0 & 0 & 0 & \\ 0 & 1 & -1 & 1 & 0 & 0 & \\ \hline 1 & 0 & -1 & 0 & 1 & 0 & \\ 1 & 1 & 1 & -1 & -1 & 1 & \\ \hline x_0 & x_1 & & & & & \end{array}$$

(b) Level-2 CFLOBDD for the 4 × 4 inverse Reed-Muller transform matrix S_2, under the interleaved variable ordering.

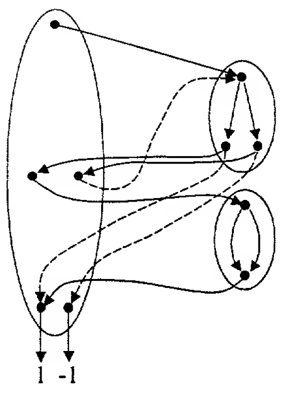


(c) Schematic drawing showing how, under the interleaved variable ordering, the level-k CFLOBDD for the $2^{2^{k-1}} \times 2^{2^{k-1}}$ inverse Reed-Muller transform matrix $S_{2^{k-1}} = S_{2^{k-2}} \otimes S_{2^{k-2}}$ is constructed from two level $k-1$ proto-CFLOBDDs.

Figure 32:

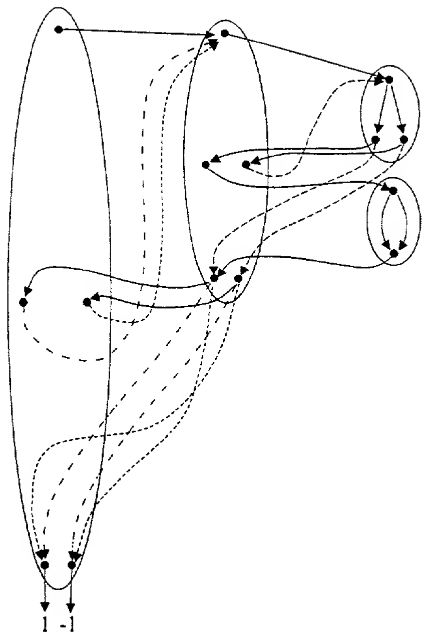
```
CFLOBDD InverseReedMullerCFLOBDD(int i) {
  return RepresentativeCFLOBDD(InverseReedMullerProtoCFLOBDD(i),[1,0,-1])
}

InternalGrouping InverseReedMullerProtoCFLOBDD(int i) {
  InternalGrouping g = new InternalGrouping(i)
  if (i == 1) {
    g.AConnection = RepresentativeForkGrouping
    g.AReturnTuple = [1,2]
    g.numberOfBConnections = 2
    g.BConnections[1] = RepresentativeForkGrouping
    g.BReturnTuples[1] = [1,2]
    g.BConnections[2] = RepresentativeForkGrouping
    g.BReturnTuples[2] = [3,1]
  }
  else {
    g.AConnection = InverseReedMullerProtoCFLOBDD(i-1)
    g.AReturnTuple = [1,2,3]
    g.numberOfBConnections = 3
    g.BConnections[1] = g.AConnection
    g.BReturnTuples[1] = [1,2,3]
    g.BConnections[2] = NoDistinctionProtoCFLOBDD(i-1)
    g.BReturnTuples[2] = [2]
    g.BConnections[3] = g.AConnection
    g.BReturnTuples[3] = [3,2,1]
  }
  g.numberOfExits = 3
  return RepresentativeGrouping(g)
}
```

Figure 33:



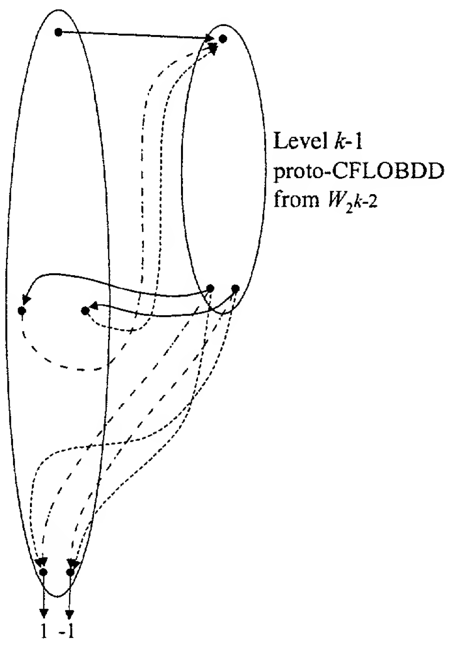
$$W_1 = \begin{array}{cc|cc|c} & & 0 & 1 & y_0 \\ \hline & 0 & 1 & 1 & \\ & 1 & 1 & -1 & \\ \hline & x_0 & & & \end{array}$$

(a) Level-1 CFLOBDD for the 2×2 Walsh transform matrix W_1, under the interleaved variable ordering.



$$W_2 = \begin{array}{cc|cc|cc|c} & & 0 & 0 & 1 & 1 & y_0 \\ & & 0 & 1 & 0 & 1 & y_1 \\ \hline 0 & 0 & 1 & 1 & 1 & 1 & \\ 0 & 1 & 1 & -1 & 1 & -1 & \\ \hline 1 & 0 & 1 & 1 & -1 & -1 & \\ 1 & 1 & 1 & -1 & -1 & 1 & \\ \hline x_0 & x_1 & & & & & \end{array}$$

(b) Level-2 CFLOBDD for the 4 × 4 Walsh transform matrix W_2, under the interleaved variable ordering.

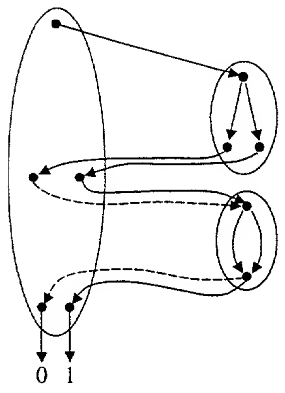


(c) Schematic drawing showing how, under the interleaved variable ordering, the level-k CFLOBDD for the $2^{2^{k-1}} \times 2^{2^{k-1}}$ Walsh transform matrix $W_{2^{k-1}} = W_{2^{k-2}} \otimes W_{2^{k-2}}$ is constructed from a level $k-1$ proto-CFLOBDD.

Figure 34:

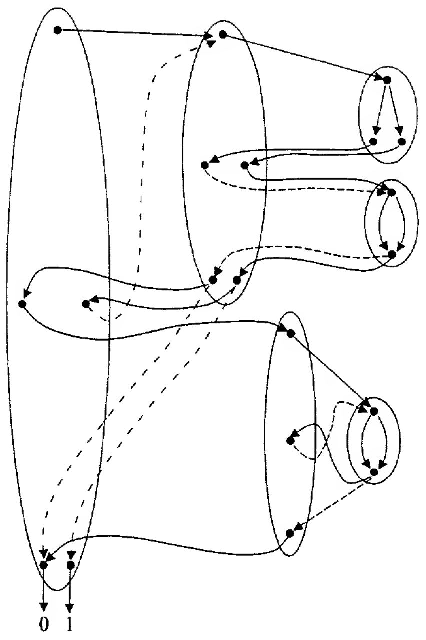
```
CFLOBDD WalshCFLOBDD(int i) {
  return RepresentativeCFLOBDD(WalshProtoCFLOBDD(i),[1,-1])
}

InternalGrouping WalshProtoCFLOBDD(int i) {
  InternalGrouping g = new InternalGrouping(i)
  if (i == 1) {
    g.AConnection = RepresentativeForkGrouping
    g.AReturnTuple = [1,2]
    g.numberOfBConnections = 2
    g.BConnections[1] = RepresentativeDontCareGrouping
    g.BReturnTuples[1] = [1]
    g.BConnections[2] = RepresentativeForkGrouping
    g.BReturnTuples[2] = [1,2]
  }
  else {
    g.AConnection = WalshProtoCFLOBDD(i-1)
    g.AReturnTuple = [1,2]
    g.numberOfBConnections = 2
    g.BConnections[1] = g.AConnection
    g.BReturnTuples[1] = [1,2]
    g.BConnections[2] = g.AConnection
    g.BReturnTuples[2] = [2,1]
  }
  g.numberOfExits = 2
  return RepresentativeGrouping(g)
}
```

Figure 35:



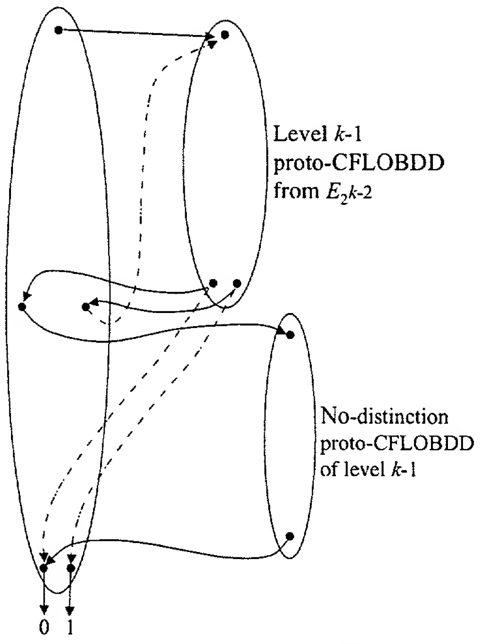
$$E_1 = \begin{array}{c|cc|l} & 0 & 1 & y_0 \\ \hline 0 & 0 & 0 & \\ 1 & 1 & 1 & \\ \hline x_0 & & & \end{array}$$

(a) Level-1 CFLOBDD for the 2 × 2 matrix E_1, under the interleaved variable ordering.



$$E_2 = \begin{array}{cc|cc|cc|l} & & 0 & 0 & 1 & 1 & y_0 \\ & & 0 & 1 & 0 & 1 & y_1 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & \\ 0 & 1 & 0 & 0 & 0 & 0 & \\ \hline 1 & 0 & 0 & 0 & 0 & 0 & \\ 1 & 1 & 1 & 1 & 1 & 1 & \\ \hline x_0 & x_1 & & & & & \end{array}$$

(b) Level-2 CFLOBDD for the 4 × 4 matrix E_2, under the interleaved variable ordering.

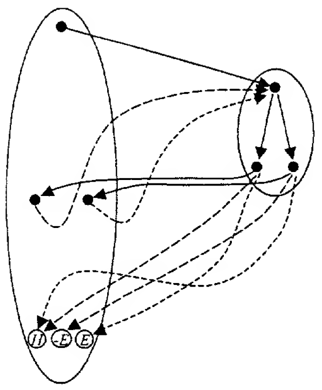

Level k-1 proto-CFLOBDD from $E_{2^{k-2}}$
No-distinction proto-CFLOBDD of level k-1
0
1

(c) Schematic drawing showing how, under the interleaved variable ordering, the level-k CFLOBDD for the $2^{2^{k-1}} \times 2^{2^{k-1}}$ matrix $E_{2^{k-1}} = E_{2^{k-2}} \otimes E_{2^{k-2}}$ is constructed from two level $k-1$ proto-CFLOBDDs.

Figure 36:

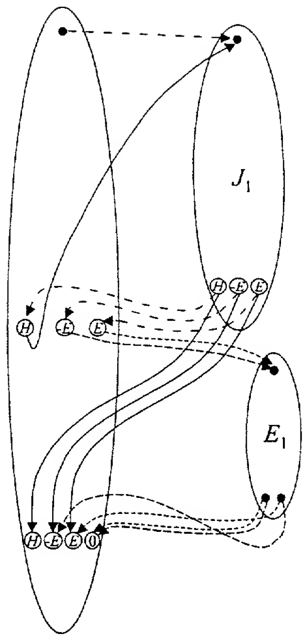
```
CFLOBDD ECFLOBDD(int i) {
  return RepresentativeCFLOBDD(EProtoCFLOBDD(i),[0,1])
}

InternalGrouping EProtoCFLOBDD(int i) {
  InternalGrouping g = new InternalGrouping(i)
  if (i == 1) {
    g.AConnection = RepresentativeForkGrouping
    g.AReturnTuple = [1,2]
    g.numberOfBConnections = 2
    g.BConnections[1] = RepresentativeDontCareGrouping
    g.BReturnTuples[1] = [1]
    g.BConnections[2] = RepresentativeDontCareGrouping
    g.BReturnTuples[2] = [2]
  }
  else {
    g.AConnection = EProtoCFLOBDD(i-1)
    g.AReturnTuple = [1,2]
    g.numberOfBConnections = 2
    g.BConnections[1] = NoDistinctionProtoCFLOBDD(i-1)
    g.BReturnTuples[1] = [1]
    g.BConnections[2] = g.AConnection
    g.BReturnTuples[2] = [1,2]
  }
  g.numberOfExits = 2
  return RepresentativeGrouping(g)
}
```

Figure 37:



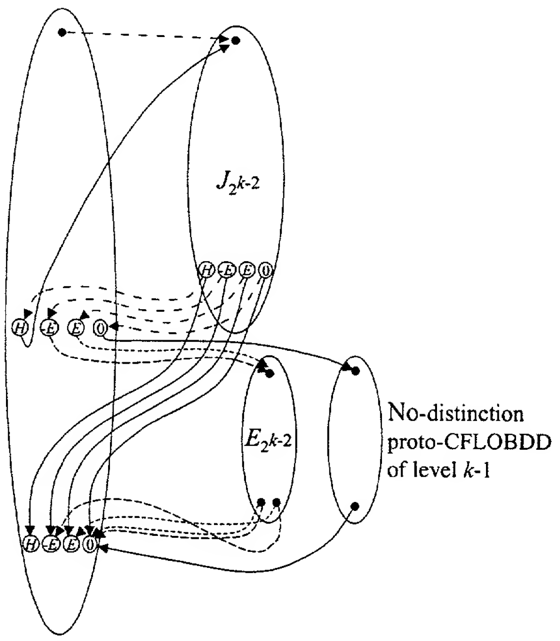
$$J_1 = \begin{array}{cc|cc|}
 & & 0 & 1 & y_0 \\ \hline
 & 0 & H & -E \\
 & 1 & E & H \\ \hline
 & x_0 & &
\end{array}$$

(a) The level-1 proto-CFLOBDD J_1, and the 2×2 matrix that it represents under the interleaved variable ordering.



$$J_2 = \begin{array}{cc|cc|cc|c}
 & & 0 & 0 & 1 & 1 & y_0 \\
 & & 0 & 1 & 0 & 1 & y_1 \\ \hline
0 & 0 & H & -E & 0 & 0 \\
0 & 1 & E & H & -E & -E \\ \hline
1 & 0 & 0 & 0 & H & -E \\
1 & 1 & E & E & E & H \\ \hline
x_0 & x_1 & & & &
\end{array}$$

(b) The level-2 proto-CFLOBDD J_2, and the 4×4 matrix that it represents under the interleaved variable ordering.

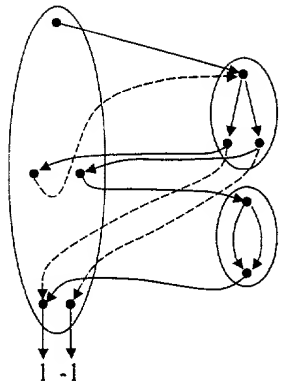


(c) Schematic drawing showing how the level-k proto-CFLOBDD $J_{2^{k-1}}$, for $k \geq 3$, is constructed from three level $k-1$ proto-CFLOBDDs.

Figure 38:

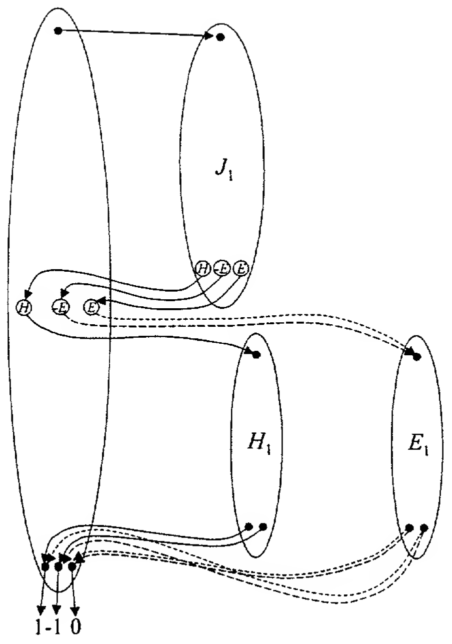
```
InternalGrouping JProtoCFLOBDD(int i) {
  InternalGrouping g = new InternalGrouping(i)
  if (i == 1) {
    g.AConnection = RepresentativeForkGrouping
    g.AReturnTuple = [1,2]
    g.numberOfBConnections = 2
    g.BConnections[1] = RepresentativeForkGrouping
    g.BReturnTuples[1] = [1,2]
    g.BConnections[2] = RepresentativeForkGrouping
    g.BReturnTuples[2] = [3,1]
    g.numberOfExits = 3
  }
  else if (i == 2) {
    g.AConnection = JProtoCFLOBDD(1)
    g.AReturnTuple = [1,2,3]
    g.numberOfBConnections = 3
    g.BConnections[1] = g.AConnection
    g.BReturnTuples[1] = [1,2,3]
    g.BConnections[2] = EProtoCFLOBDD(1)
    g.BReturnTuples[2] = [4,2]
    g.BConnections[3] = g.BConnections[2]
    g.BReturnTuples[3] = [4,3]
    g.numberOfExits = 4
  }
  else {
    g.AConnection = JProtoCFLOBDD(i-1)
    g.AReturnTuple = [1,2,3,4]
    g.numberOfBConnections = 4
    g.BConnections[1] = g.AConnection
    g.BReturnTuples[1] = [1,2,3,4]
    g.BConnections[2] = EProtoCFLOBDD(i-1)
    g.BReturnTuples[2] = [4,2]
    g.BConnections[3] = g.BConnections[2]
    g.BReturnTuples[3] = [4,3]
    g.BConnections[4] = NoDistinctionProtoCFLOBDD(i-1)
    g.BReturnTuples[4] = [4]
    g.numberOfExits = 4
  }
  return RepresentativeGrouping(g)
}
```

Figure 39:



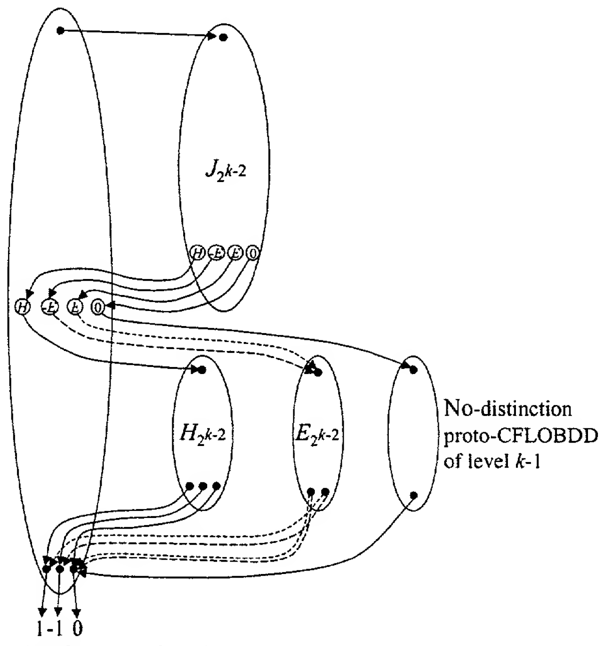
$$H_1 = \begin{array}{c|cc|l} & 0 & 1 & y_0 \\ \hline 0 & 1 & -1 & \\ 1 & 1 & 1 & \\ \hline x_0 & & & \end{array}$$

(a) Level-1 CFLOBDD for the 2 × 2 Boolean Haar Wavelet transform matrix H_1, under the interleaved variable ordering.



$$H_2 = \begin{array}{cc|cc|cc|l} & & 0 & 0 & 1 & 1 & y_0 \\ & & 0 & 1 & 0 & 1 & y_1 \\ \hline 0 & 0 & 1 & -1 & 0 & 0 & \\ 0 & 1 & 1 & 1 & -1 & -1 & \\ \hline 1 & 0 & 0 & 0 & 1 & -1 & \\ 1 & 1 & 1 & 1 & 1 & 1 & \\ \hline x_0 & x_1 & & & & & \end{array}$$

(b) Level-2 CFLOBDD for the 4 × 4 Boolean Haar Wavelet transform matrix H_2, under the interleaved variable ordering.



(c) Schematic drawing showing how, under the interleaved variable ordering, the level-k CFLOBDD for the $2^{2^{k-1}} \times 2^{2^{k-1}}$ Boolean Haar Wavelet transform matrix $H_{2^{k-1}}$ is constructed from four level $k-1$ proto-CFLOBDDs.

Figure 40:

```
CFLOBDD HaarCFLOBDD(int i) {
  if (i == 1) return RepresentativeCFLOBDD(HaarProtoCFLOBDD(1),[1,-1])
  else return RepresentativeCFLOBDD(HaarProtoCFLOBDD(i),[1,-1,0])
}

InternalGrouping HaarProtoCFLOBDD(int i) {
  InternalGrouping g = new InternalGrouping(i)
  if (i == 1) {
    g.AConnection = RepresentativeForkGrouping
    g.AReturnTuple = [1,2]
    g.numberOfBConnections = 2
    g.BConnections[1] = RepresentativeForkGrouping
    g.BReturnTuples[1] = [1,2]
    g.BConnections[2] = RepresentativeDontCareGrouping
    g.BReturnTuples[2] = [1]
    g.numberOfExits = 2
  }
  else if (i == 2) {
    g.AConnection = JProtoCFLOBDD(1)
    g.AReturnTuple = [1,2,3]
    g.numberOfBConnections = 3
    g.BConnections[1] = HaarProtoCFLOBDD(1)
    g.BReturnTuples[1] = [1,2]
    g.BConnections[2] = EProtoCFLOBDD(1)
    g.BReturnTuples[2] = [3,2]
    g.BConnections[3] = g.BConnections[2]
    g.BReturnTuples[3] = [3,1]
    g.numberOfExits = 3
  }
  else {
    g.AConnection = JProtoCFLOBDD(i-1)
    g.AReturnTuple = [1,2,3,4]
    g.numberOfBConnections = 4
    g.BConnections[1] = HaarProtoCFLOBDD(i-1)
    g.BReturnTuples[1] = [1,2,3]
    g.BConnections[2] = EProtoCFLOBDD(i-1)
    g.BReturnTuples[2] = [3,2]
    g.BConnections[3] = g.BConnections[2]
    g.BReturnTuples[3] = [3,1]
    g.BConnections[4] = NoDistinctionProtoCFLOBDD(i-1)
    g.BReturnTuples[4] = [3]
    g.numberOfExits = 3
  }
  return RepresentativeGrouping(g)
}
```

Figure 41:

```
enum VisitState { FirstVisit, SecondVisit, ThirdVisit, Restart }

class TraverseState {
  grouping: Grouping
  visitState: VisitState
  index: int // only relevant for when visitState == ThirdVisit
}

ValueTuple UncompressCFLOBDD(CFLOBDD C) {
  ValueTuple ans = []
  Tuple of TraverseState S // Traversal stack; right end is top of stack
  Tuple of (Tuple of TraverseState) T // Stack of snapshots of S
  int exitVertexIndex

  T = [[new TraverseState(C.grouping,FirstVisit,_)]]
  while (T != []) {
    [T,S] = SplitOnLast(T) // S = last element of T; T = all of T but the last element
    while (S != []) {
      TraverseState [S,ts] = SplitOnLast(S) // ts = last element of S; S = all but last
      if (ts.grouping == RepresentativeForkGrouping) {
        if (ts.visitState == FirstVisit) { // follow the left branch
          T = T || (S || new TraverseState(ts.grouping,Restart,_)) // record snapshot
          exitVertexIndex = 1
        }
        else { // ts.visitState == Restart; this time follow the right branch
          exitVertexIndex = 2
        }
      }
      else if (ts.grouping == RepresentativeDontCareGrouping) {
        if (ts.visitState == FirstVisit) { // follow the left branch
          T = T || (S || new TraverseState(ts.grouping,Restart,_)) // record snapshot
          exitVertexIndex = 1
        }
        else { // ts.visitState == Restart; this time follow the right branch
          exitVertexIndex = 1
        }
      }
      else { // Must be a CFLOBDDInternalGrouping
        CFLOBDDInternalGrouping g = (CFLOBDDInternalGrouping)ts.grouping
        if (ts.visitState == FirstVisit) {
          S = S || new TraverseState(g,SecondVisit,_)
          S = S || new TraverseState(g.AConnection,FirstVisit,_)
        }
        else if (ts.visitState == SecondVisit) {
          int i = g.AReturnTuple[exitVertexIndex]
          S = S || new TraverseState(g,ThirdVisit,i)
          S = S || new TraverseState(g.BConnection[i],FirstVisit,_)
        }
        else { // ts.visitState == ThirdVisit
          exitVertexIndex = g.BReturnTuples[ts.index](exitVertexIndex)
        }
      } // while (S != []) loop
    } // while (T != []) loop
    ans = ans || C.valueTuple[exitVertexIndex] // Finished processing one assignment
  }
  return ans
}
```

Figure 42: